



Department of Computer Science and Engineering

Course Code: CSE341	Credits: 1.5
Course Name: Microprocessors	Semester: Spring'25

Lab 04

Flow control instructions and Looping structures

I. Topic Overview:

Alongside the capability of making decisions, any functional program should also have a mechanism to repeat sections of code. In this lab, students will familiarize themselves with the loop instructions to achieve that. Alongside jump, the loop instruction is also used to transfer control to another part of the program. The students will then learn to implement different looping structures. This application will make it much easier to convert a pseudo code algorithm to assembly code.

II. Lesson Fit:

There are prerequisites to this lab. Students must have a basic idea on the following concepts:

- Jump instruction
- Some basic operations such as MOV, ADD, SUB, MUL and DIV
- Basic I/O operations
- Character encoding using ASCII

III. Learning Outcome:

After this lecture, the students will be able to:

- Control flow of the program
- Use loops to avoid repetition

IV. Anticipated Challenges and Possible Solutions

- a. Students may find it difficult to visualize how the program control flow changes when using loop/jump

Solutions:

- i. Step by step simulation
- b. Directly coding in assembly may come off as challenging

Solutions:

- i. Writing the pseudocode first then then converting it to assembly may help

V. Acceptance and Evaluation

Students will show their progress as they complete each problem. They will be marked according to their class performance. There may be students who might not be able to finish all the tasks, they will submit them later and give a viva to get their performance mark. A deduction of 30% marks is applicable for late submission. The marks distribution is as follows:

Code: 50%

Viva: 50%

VI. Activity Detail

a. Hour: 1

Discussion: Looping Structure

A loop is a sequence of instructions that is repeated. In high level languages we have studied While Loop and For Loop. While loop depends on a condition and For is a loop structure in which the loop statements are repeated a known number of times (a count-controlled loop). Loops can also be implemented in assembly language in **2 ways**:

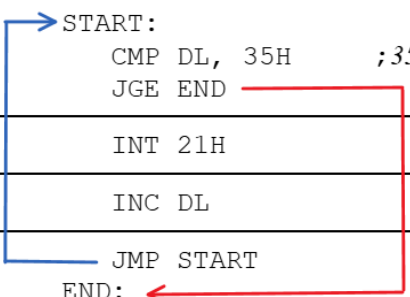
Loop Type	Description
Explicit Loop	Depends on conditions (Similar to While Loop)
Implicit Loop	Repeated a known number of times (Similar to For Loop)

1. Explicit Loop:

Explicit loops can be implemented by using compare (CMP) and jump instructions to decide whether to enter the loop or not and by using the **inc/ dec/ add/ sub** instruction for increments and decrements.

Example: Write a program to print from 0 to 4.

Line by line code conversion from Java to Assembly:

Java	Assembly
<code>int x = 0;</code>	<code>MOV AH, 2 ;single character output</code> <code>MOV DL, 30H ;emu8086 uses hexadecimal. 30h is 0</code>
<code>while (x < 5){</code>	 <code>→ START:</code> <code> CMP DL, 35H ;35h is 5</code> <code> JGE END</code>
<code> System.out.println(x);</code>	<code> INT 21H</code>
<code> x++;</code>	<code> INC DL</code>
<code>}</code>	<code> JMP START</code> <code>END: ←</code>

How it works: As we need to print from 0 to 4, we are saving 0 or 30H in DL. Then we are labeling the start of the while loop with 'START' and exit of the loop with 'END'. The loop breaking condition is $x < 5$ and we can use CMP and JGE for this. When DL is greater than or equal to 35h, we go to END and break the loop, otherwise we print the current value of DL, Increment DL and jump back to START.

2. Implicit Loop:

In this case we do not have to **check** whether the counter has reached the limit or not. This will be done automatically like For loop. Here we use **LOOP** instruction to control the flow. **CX register will be used always as the counter** in order for the LOOP instruction to execute.

Instruction Syntax: **LOOP destination_line**

Example: Write a program to print from 0 to 4.

Line by line code conversion from Java to Assembly:

Java	Assembly
int x = 0;	MOV AH, 2 ; single character output MOV DL, 30H ; emu8086 uses hexadecimal. 30h is 0
for (int i=0; i<5; i++){	MOV CX, 5 ; the bound will be in CX. (the number of times the loop will run)
System.out.println(x);	→ START: INT 21H
x++;	INC DL
}	LOOP START

NB. **CX will always start from the specified count and will always decrement by 1.** When CX = 0, the loop will break.

How it works: As we need to print from 0 to 4, we are saving 0 or 30H in DL. Then we are storing the counter 5 inside the CX register. Here, we are labeling the start of the loop with 'START' and the loop will automatically end when CX = 0. Inside the START block we are printing DL and incrementing it. When the LOOP instruction gets executed the CX gets decremented by 1 and it checks whether the CX is 0 or not.

Problem Task: Task 01-Task03 (Page 7)

b. Hour: 2

Discussion: Discuss the properties of a repeat-until loop structure.

Repeat Until loop:

Another conditional loop is the Repeat Until Loop where a loop is repeated until a condition is satisfied.

Example: You have been asked to take in characters from the user and print them until a space is pressed.

Line by line code conversion from Java to Assembly:

Java	Assembly
while (true){	→ REPEAT:
char input=scanner.next().charAt(0);	MOV AH, 1 ; single character input INT 21H ; char in AL
System.out.println(input);	MOV AH, 2 ; single character output MOV DL, AL INT 21H
if (input == ' '){	CMP AL, ' '
break; } }	JNE REPEAT

How it works: We need to implement an infinite loop as we don't know when the user will press space. At first we take an input and show it as an output inside the REPEAT block. If the user enters any character other than space, the JNE (Jump Not Equal) instruction will become True (will check flag status), the program will jump to the REPEAT label and continue the loop again. If the user presses space, the JNE instruction will become False and the program pointer will go to the next line, thus it will break the loop.

Differences Between While Loop and Repeat Until Loop:

In many situations where a conditional loop is needed, use of a WHILE loop or a REPEAT Until loop is a matter of personal preference. However, we can choose one by evaluating their advantages and disadvantages.

Feature	WHILE Loop	REPEAT Until Loop
Condition Check	Condition is checked before executing the loop body.	Condition is checked after executing the loop body.
Execution Guarantee	May not execute at all if the condition is initially false.	Always executes at least once .
Jump Operations	Has two jumps : one conditional jump at the top and one unconditional jump (JMP) at the bottom.	Has only one conditional jump at the end.
Use Case	Used when looping should occur only if a condition is true from the start.	Used when at least one iteration is required before checking the condition.

Problem Task: Task 04 – 06 (Page 7-8)

c. Hour: 3

Discussion:

Check progress while the students carry on with the rest of the tasks.

Problem Task: Task 07 (Page 8)

VII. Home Tasks: All the unfinished lab tasks.

Lab 5 Activity List

Task 01

Write an implicit loop to display a row of 80 stars.

Task 02

Write a sequence of instructions to do each of the following:

- a. Put the sum of $1 + 4 + 7 + \dots + 148$ in AX.
- b. Put the sum $100 + 95 + 90 + \dots + 5$ in a variable "sum".

Task 03

Read a five character password and overprint it by executing a carriage return and displaying five X's. You don't need to store the input characters.

Task 04

The following algorithm may be used to carry out multiplication of two positive numbers M and N by repeated addition.

Initialize a variable named "product" to 0

Start the loop

add M to "product"

decrement N

UNTIL N equals 0

End the loop

Task 05

Write a program to display the extended ASCII characters (ASCII codes 80h to FFh). Display 10 characters per line, separated by blanks. Stop after the extended characters have been displayed once.

Task 06

Write a program that will prompt the user to enter a hex digit character ("0" ... "9" or "A" ... "F"), display it on the next line in decimal, and ask the user if he or she wants to do it again. If the user types "y" or "Y", the program repeats; If the user types anything else, the program terminates. If the user enters an illegal character, prompt the user to try again

Sample input: ENTER A HEX DIGIT: 9

Sample output: IN DECIMAL IT IS 9
DO YOU WANT TO DO IT AGAIN? y

Sample input: ENTER A HEX DIGIT: C

Sample output: IN DECIMAL IT IS 12
DO YOU WANT TO DO IT AGAIN? N

Task 07

[Hard] Write a program that reads a string of capital letters, ending with a carriage return, and displays the longest sequence of consecutive alphabetically increasing capital letters read.

Sample input: ENTER A STRING OF CAPITAL LETTERS: FGHADEFGHC

Sample output: THE LONGEST CONSECUTIVELY INCREASING STRING IS:
DEFGH

Task 08

From a sequence of natural numbers 1, 2, 3, 4,, 100 add all those numbers that are divisible by the last non zero digit of your student ID and put the summation in a variable M. Add the other numbers that are not divisible by that digit and put the summation in another variable N.

Task 09

Write an assembly program that takes an integer input from the user (ranging from 1 to 9) and uses nested loops to generate the following pattern, where the number of rows corresponds to the user's input:

Sample input:

5

Sample Output:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Task 10

Write an assembly program that prompts the user to enter four single-character inputs (e.g., '1', '9', '9', '6') representing a year, converts them into a 4-digit integer using loop, and checks if the year is a leap year or not. The program should output either "Leap year" or "Not leap year" based on the leap year rules

- **Case 1:** It should be divisible by 4 ($\text{inputYear} \% 4 == 0$).
- **Case 2:** Alternatively, if the year is divisible by 400 ($\text{inputYear} \% 400 == 0$) but not by 100 ($\text{inputYear} \% 100 != 0$), it is also a leap year.

Sample input

1996

Sample Output

leap year

Sample input

2019

Sample Output

not leap year